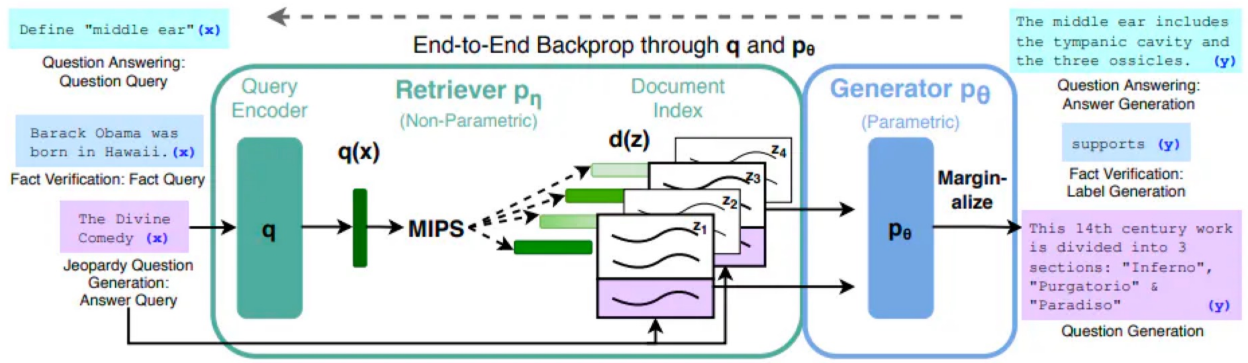


Figure 1: Overview of our approach. We combine a pre-trained retriever (*Query Encoder* + *Document Index*) with a pre-trained seq2seq model (*Generator*) and fine-tune end-to-end. For query x, we use Maximum Inner Product Search (MIPS) to find the top-K documents z_i. For final prediction y, we treat z as a latent variable and marginalize over seq2seq predictions given different documents.

Retrieval Augmented Generation (RAG)

Outline

- Intro into Large Language Models (LLMs)
- RAG pipeline
- Embedding vectors
  - Sentence BERT
- Vector databases
  - Clustering algorithms (HNSW)

Prerequisites

- Structure of the Transformer model
- How attention mechanism works
- BERT
  - MLM task
  - [CLS] token

What is a Language Model?

A Language Model is a probabilistic model that assigns probabilities to sequences of words. In practice, a Language Model allows us to compute the following:

$$P\left[\ \underbrace{\text{“China”}}_{\textbf{Next Token}}\ \middle|\ \underbrace{\text{“Shanghai is a city in”}}_{\textbf{Prompt}}\ \right]$$

We usually train a neural network to predict these probabilities. A neural network trained on a large corpora of text is known as a Large Language Model (LLM).

How do we train and perform inference on a Language Model?

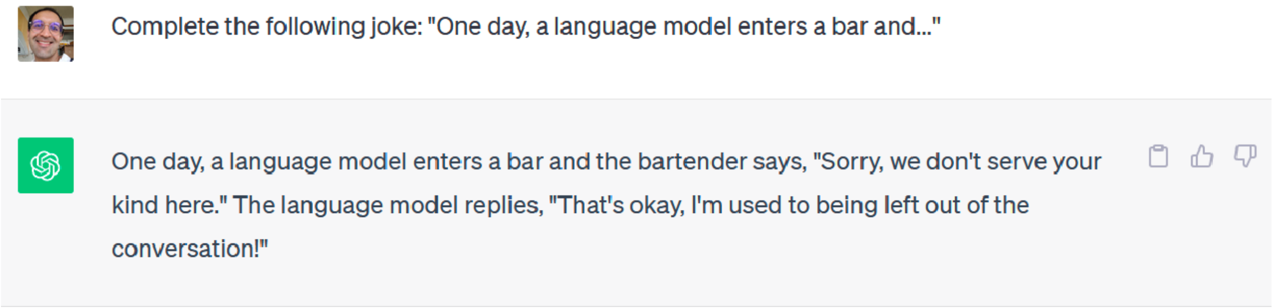
Training

A Language Model is trained on a corpora of text, that is, a large collection of documents. Often, Language Models are trained on entire Wikipedia and millions of web pages. This allows the Language Model to acquire as much knowledge as possible.

We usually train Transformer-based neural networks as Language Models.

Inference

To perform inference, we construct a prompt and let the Language Model generate the rest by iteratively adding tokens.



You Are What You Eat

A language model can only output text and information that it was trained upon. This means that if we train a language model only on English content, it is very likely that it won't be able to output Japanese or French. To teach new concepts, we need to fine-tune the model.

The Cons of Fine-tuning

- It can be expensive
- The number of parameters of the model may not be sufficient to capture all the knowledge we want to teach it. That’s why LLaMA was introduced with 7B, 13B and 70B parameters.
- Fine-tuning is not additive. It may replace existing knowledge of the model with new knowledge. For example, a language model trained on English that is (heavily) fine-tuned on Japanese may “forget” some English.

Prompt Engineering to the rescue!

It is possible to “teach” a language model how to perform a new task by playing with the prompt. For example, by using “few-shot” prompting. The following is an example:

奥利奥 is a cat that likes to play tricks on his friend Umar by replacing all the names in everything he writes with "meow".

For example:

Umar writes: "Bob runs a YouTube channel."
奥利奥 modifies it to: "Meow runs a YouTube channel."

Umar writes: "Alice likes to play with his friend Bob"
奥利奥 modifies it to:

"Meow likes to play with his friend Meow."

Question Answering (QA) with Prompt Engineering

Prompt

Instructions

You're an assistant trained to answer questions using the given context.

Context

Context:
"The engine powering Grok is Grok-1, our frontier LLM, which we developed over the last four months. Grok-1 has gone through many iterations over this span of time.
After announcing xAI, we trained a prototype LLM (Grok-0) with 33 billion parameters. This early model approaches LLaMA 2 (70B) capabilities on standard LM benchmarks but uses only half of its training resources. In the last two months, we have made significant improvements in reasoning and coding capabilities leading up to Grok-1, a state-of-the-art language model that is significantly more powerful, achieving 63.2% on the HumanEval coding task and 73% on MMLU.
To understand the capability improvements we made with Grok-1, we have conducted a series of evaluations using a few standard machine learning benchmarks designed to measure math and reasoning abilities.

GSM8k: Middle school math word problems, (Cobbe et al. 2021), using the chain-of-thought prompt.
MMLU: Multidisciplinary multiple choice questions, (Hendrycks et al. 2021), provided 5-shot in-context examples.
HumanEval: Python code completion task, (Chen et al. 2021), zero-shot evaluated for pass@1.
MATH: Middle school and high school mathematics problems written in LaTeX, (Hendrycks et al. 2021), prompted with a fixed 4-shot prompt."

Question

Answer the following question: "How many parameters are there in Grok-0?"

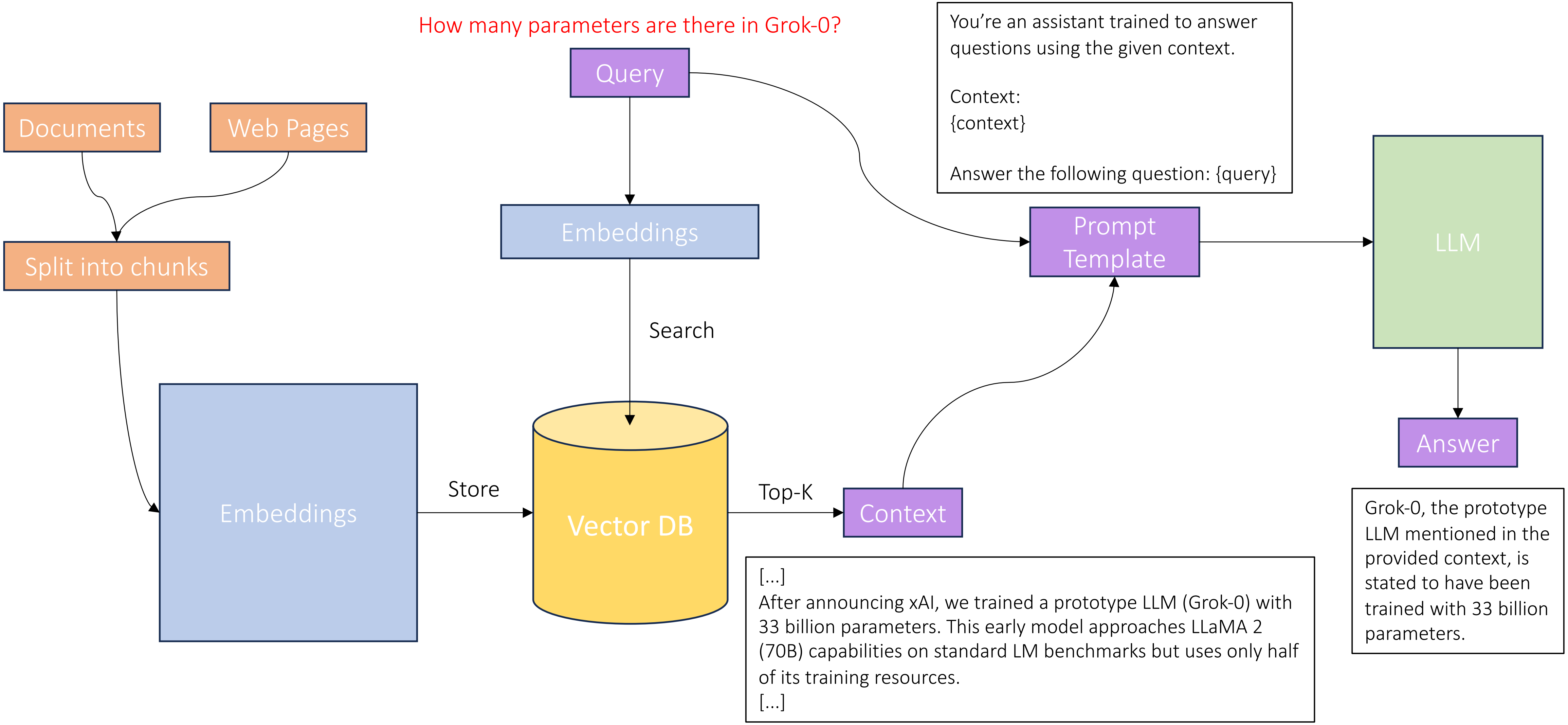
Answer

Grok-0, the prototype LLM mentioned in the provided context, is stated to have been trained with 33 billion parameters.

The Pros of Fine-tuning

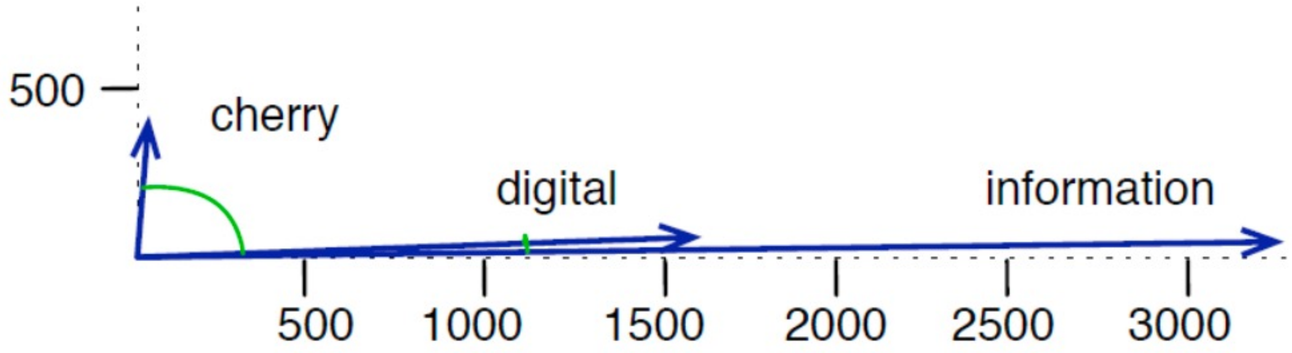
- Higher quality results compared to prompt engineering
- Smaller context size (input size) during inference since we don’t need to include the context and instructions

QA with Retrieval Augmented Generation



Why do we use vectors to represent words?

Given the words "**cherry**", "**digital**" and "**information**", if we represent the embedding vectors using only 2 dimensions (X, Y) and we plot them, we hope to see something like this: the angle between words with similar meaning is small, while the angle between words with different meaning is big. So, the embeddings "capture" the meaning of words they represent by projecting them into a high-dimensional space.



We commonly use the **cosine similarity**, which is based on the **dot product** between the two vectors.

Word Embeddings: The Ideas

- Words that are synonymous tend to occur in the same context (surrounded by the same words).
  - For example, the word “teacher” and “professor” usually occur in the same context and surrounded by the same words like ”school”, “university”, “exam”, “lecture”, “course”, etc.
- The inverse can also be true: words that occur in the same context tend to have similar meanings. This is known as **distributional hypothesis**.
- This means that to capture the meaning of the word, we also need to have access to its context (the word surrounding it).
- This is why we employ the Self-Attention mechanism in the Transformer model to capture contextual information of every token. The Self-Attention mechanism relates every token to all the other tokens in the sentence.

Word Embeddings: The Cloze Task

- Imagine I give you the following sentence:
  Rome is the ________ of Italy, which is why it hosts many government buildings.
  Can you tell me the missing word?
- Of course! The missing word is "*capital*", because by looking at the rest of the sentence, it is the one that makes the most sense.
- This is how we train BERT: we want the Self-Attention mechanism to relate all the input tokens with each other, so that BERT has enough information about the "context" of the missing word to predict it.

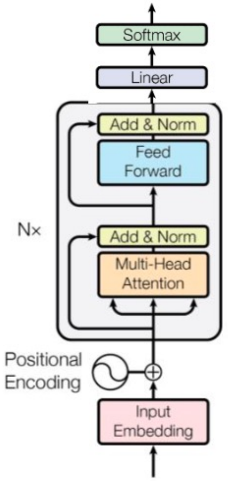
How do we train embedding vectors in BERT?

Target (1 token) capital

Loss → Run **backpropagation** to update weights

Output (14 tokens)

TK1	TK2	TK3	TK4	TK5	TK6	TK7	TK8	TK9	TK10	TK11	TK12	TK13	TK14

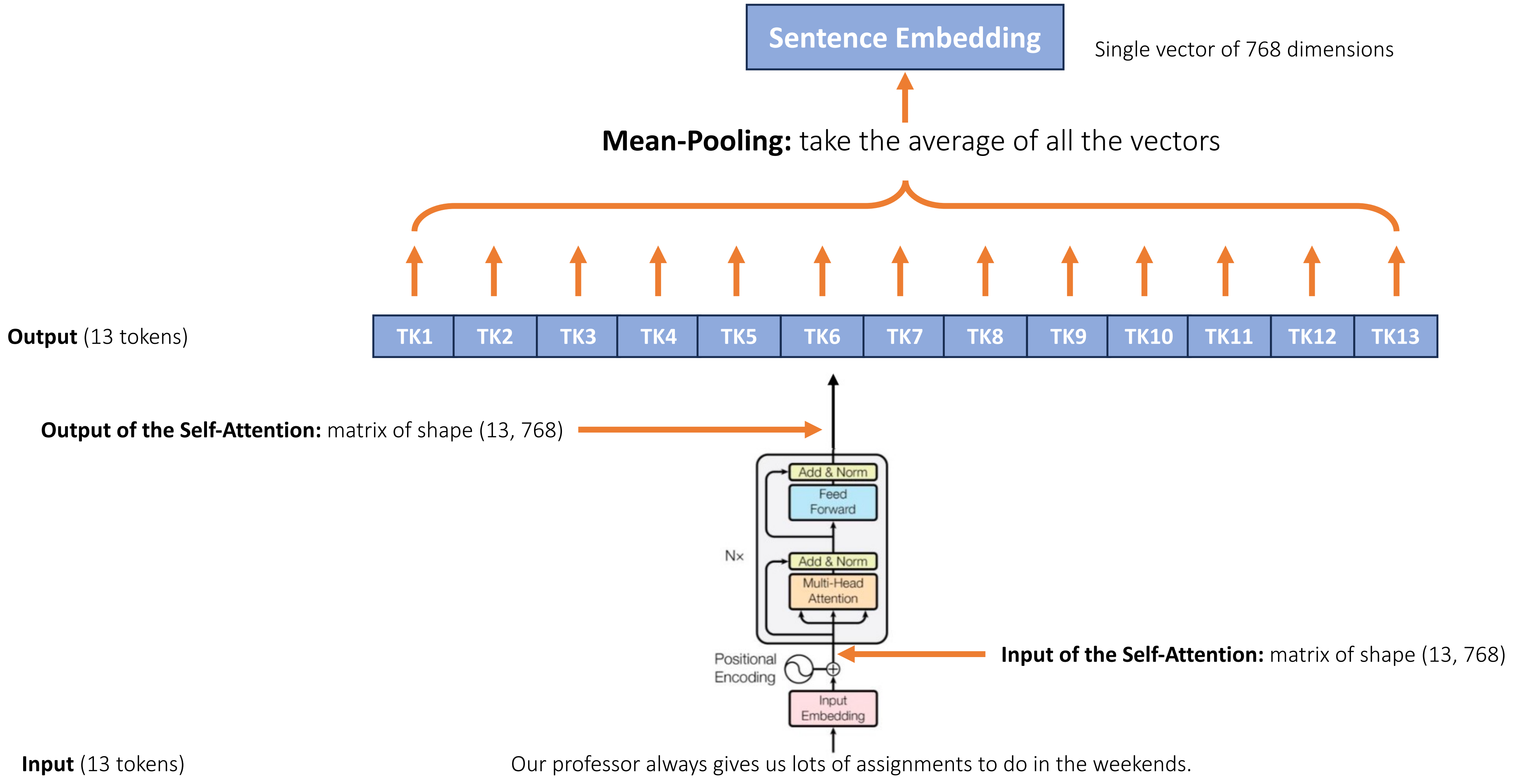


Input (14 tokens) Rome is the **[mask]** of Italy, which is why it hosts many government buildings.

Sentence Embeddings

- We can use Self-Attention mechanism also to capture the ”meaning” of an entire sentence
- We can use a pre-trained BERT model to produce embeddings of entire sentences. Let’s see how

Sentence Embeddings with BERT



Sentence Embeddings: Similarity Comparison

- How can we compare Sentence Embeddings to see if two sentences have similar “meaning”? We could use cosine similarity which measures the cosine of the angle between two vectors. A small angle results in a high cosine similarity score.

$$\text{cosine similarity} = S_C(A, B) \coloneqq \cos(\theta) = \frac{\mathrm{A} \cdot \mathrm{B}}{\|\mathrm{A}\|\|\mathrm{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

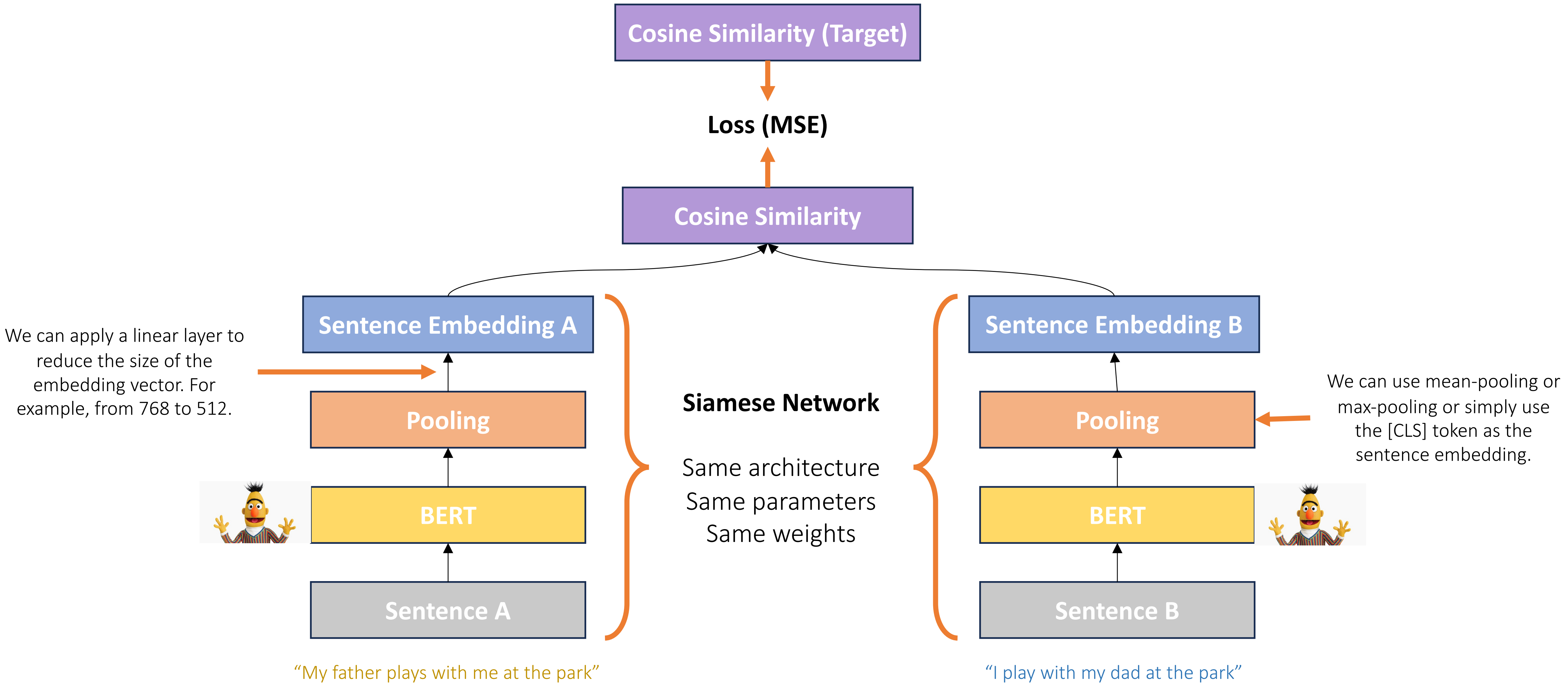
- **But there’s a problem.** Nobody told BERT that the embeddings it produces should be comparable with the cosine similarity. That is, two similar sentences should be represented by vectors pointing to the same direction in space. How can we teach BERT to produce embeddings that can be compared with a similarity function of our choice?

Introducing Sentence BERT

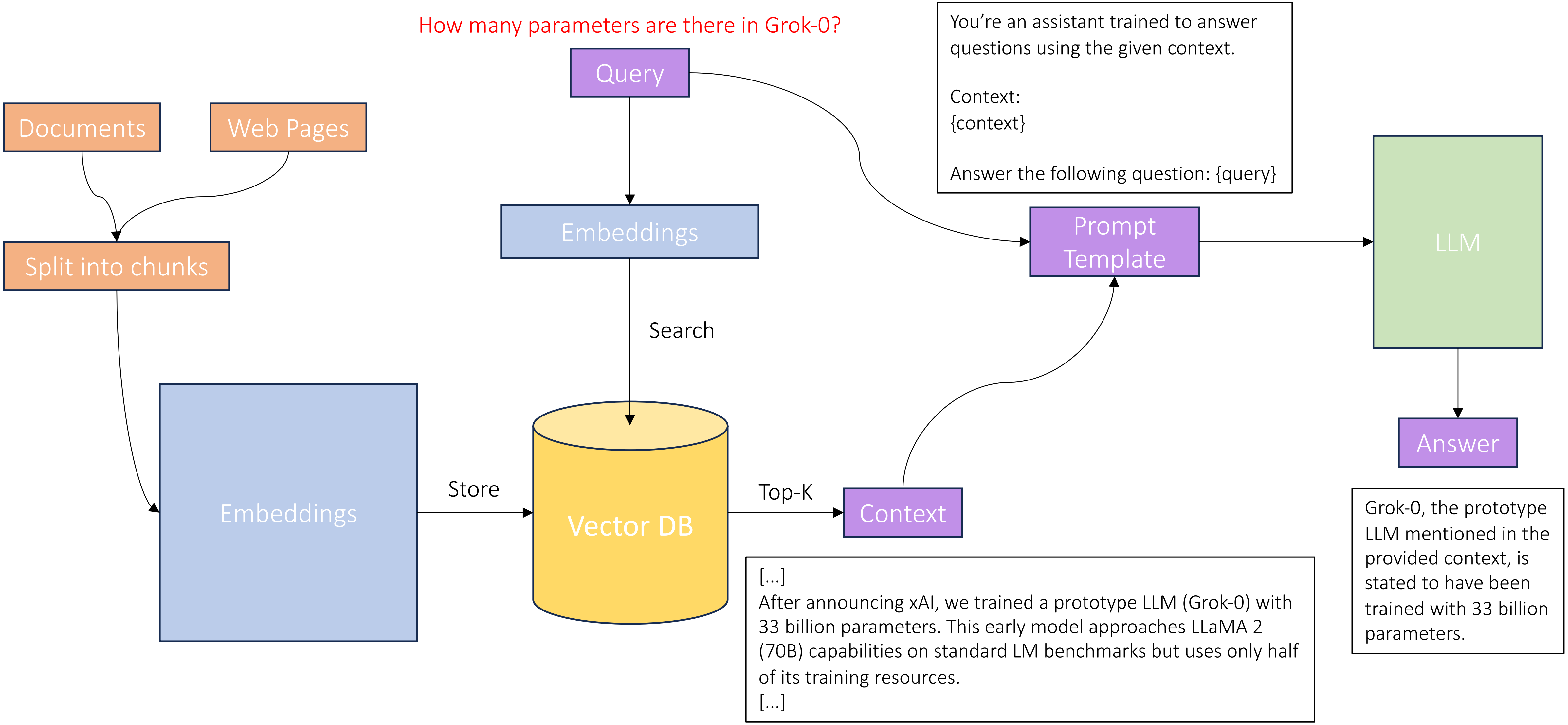
Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks

Nils Reimers and Iryna Gurevych
Ubiquitous Knowledge Processing Lab (UKP-TUDA)
Department of Computer Science, Technische Universität Darmstadt
www.ukp.tu-darmstadt.de

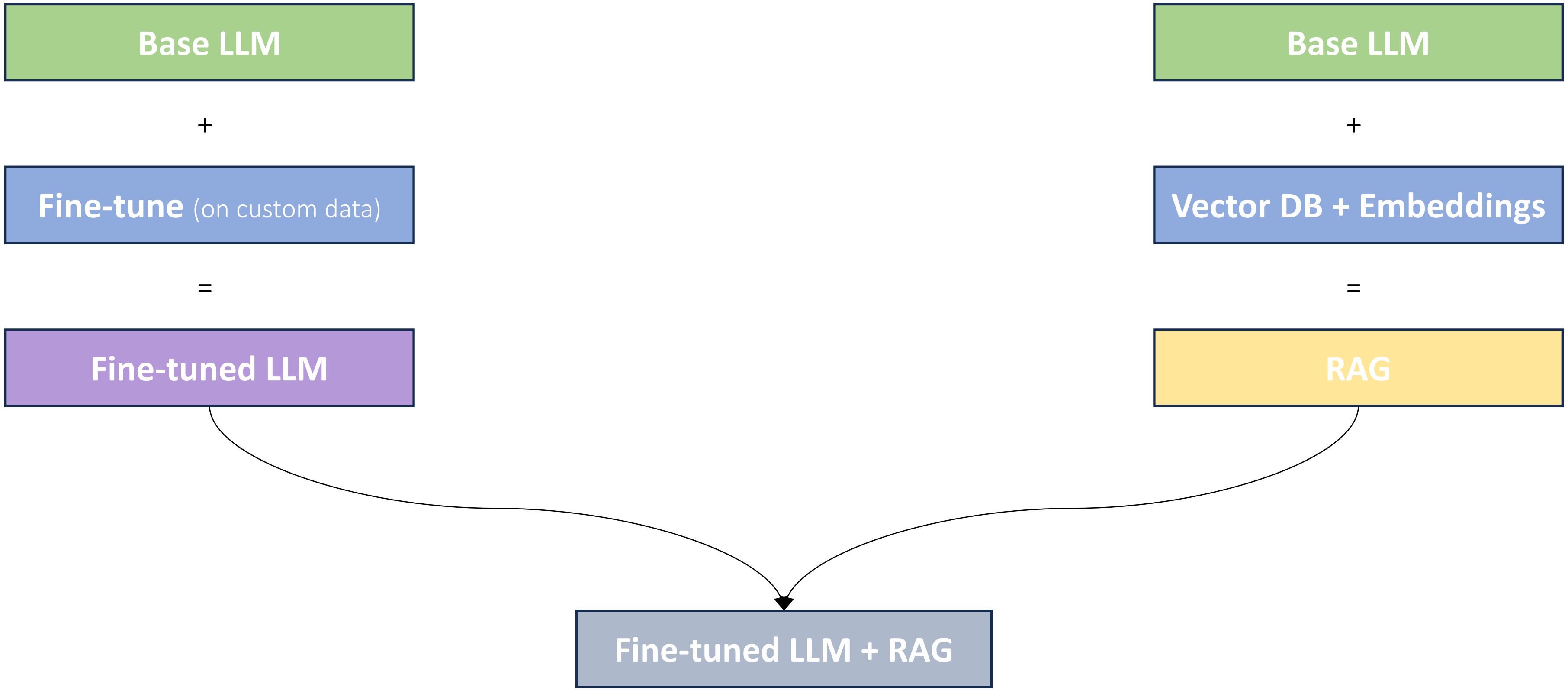
Sentence BERT: Architecture



QA with Retrieval Augmented Generation

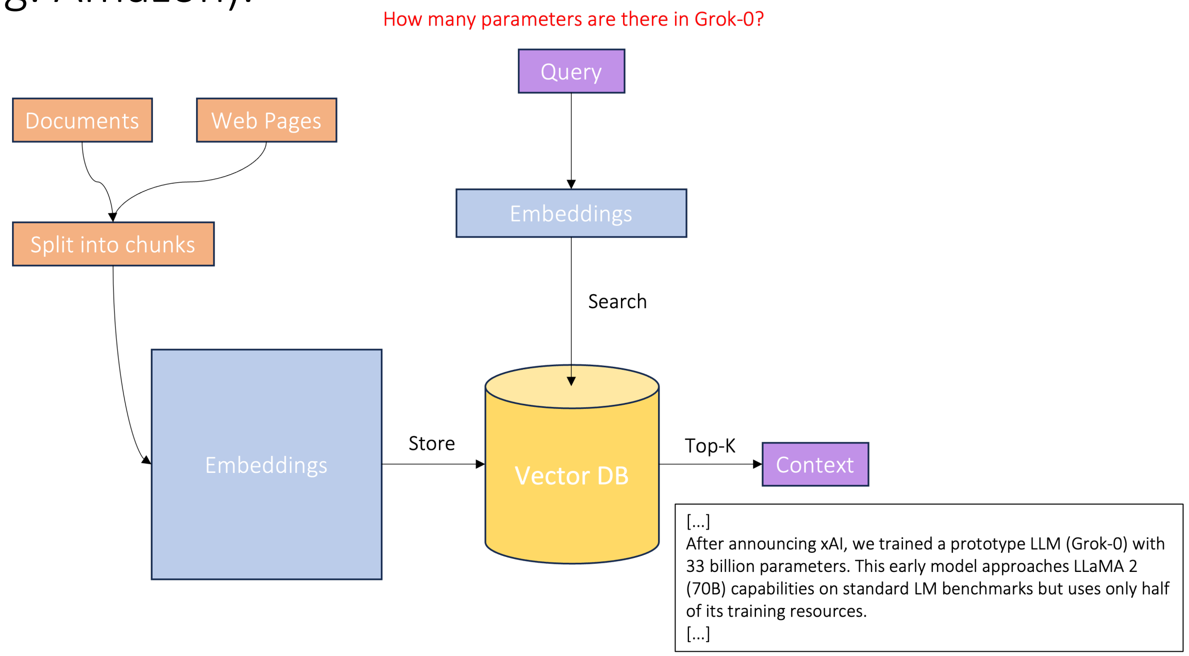


Strategies to teach new concepts to LLMs



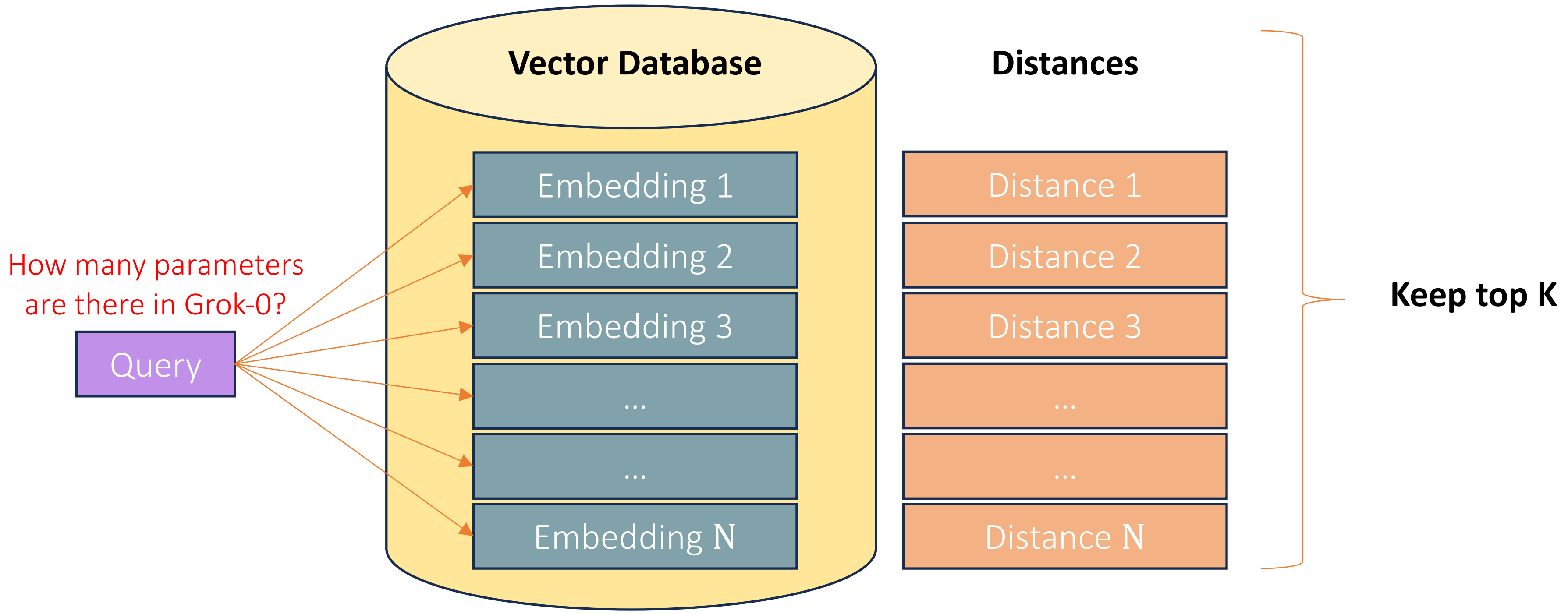
Vector DB: Introduction

A vector database stores vector of fixed dimensions (called embeddings) such that we can then query the database to find all the embeddings that are closest (most similar) to a given query vector using a distance metric, which is usually the cosine similarity, but we can also use the Euclidean distance. The database uses a variant of K-Nearest Neighbors (K-NN) algorithm or another similarity search algorithm. Vector DBs are also used for finding similar songs (e.g. Spotify), images (e.g. Google Photos) or products (e.g. Amazon).



K-NN: A Naïve Approach

Imagine we want to search for the query in our database: a simple way would be comparing the query with all vectors, sorting them by distance, and keeping the top K.



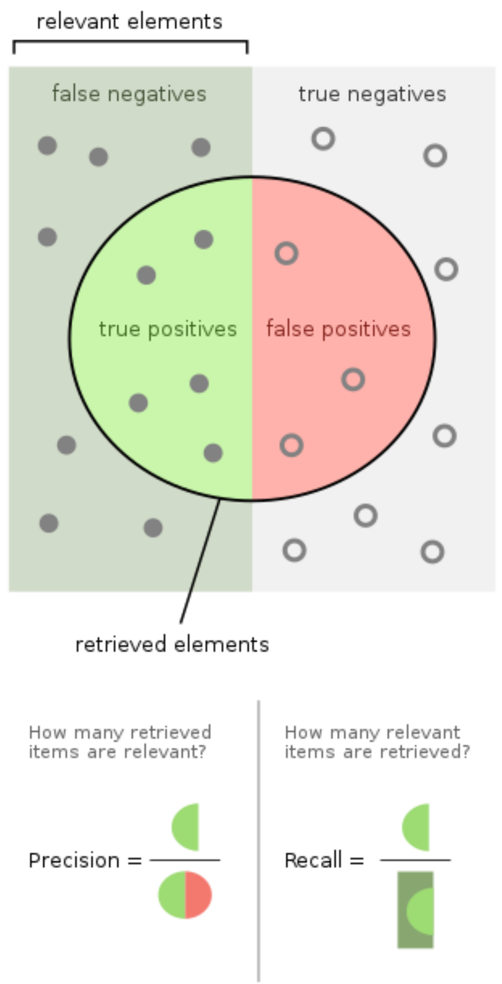
If there are N embedding vectors and each has N dimensions, the computational complexity is in the order of $O(N * D)$ which is **too slow!**

Similarity Search: Let's trade Precision for Speed

The naïve approach we used before, always produces accurate results, since it compares the query with all the stored vectors, but what if we reduced the number of comparison, but still obtain accurate results with high probability?

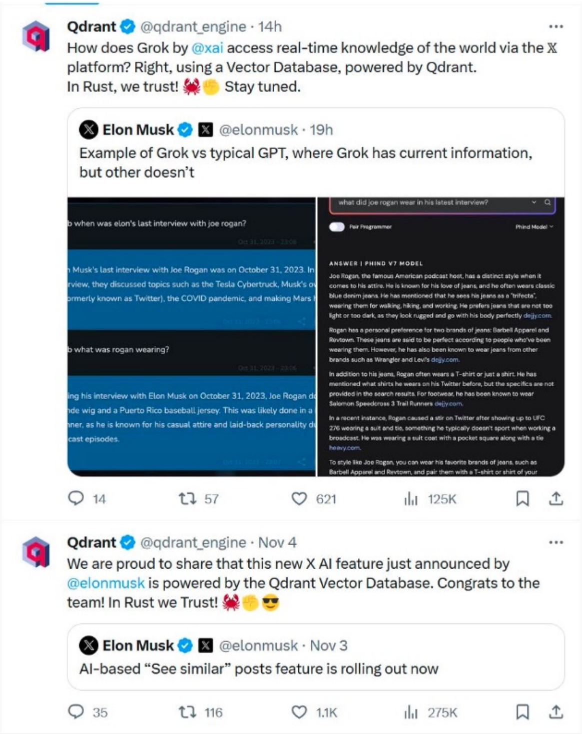
The metric we usually care about in Similarity Search is recall.

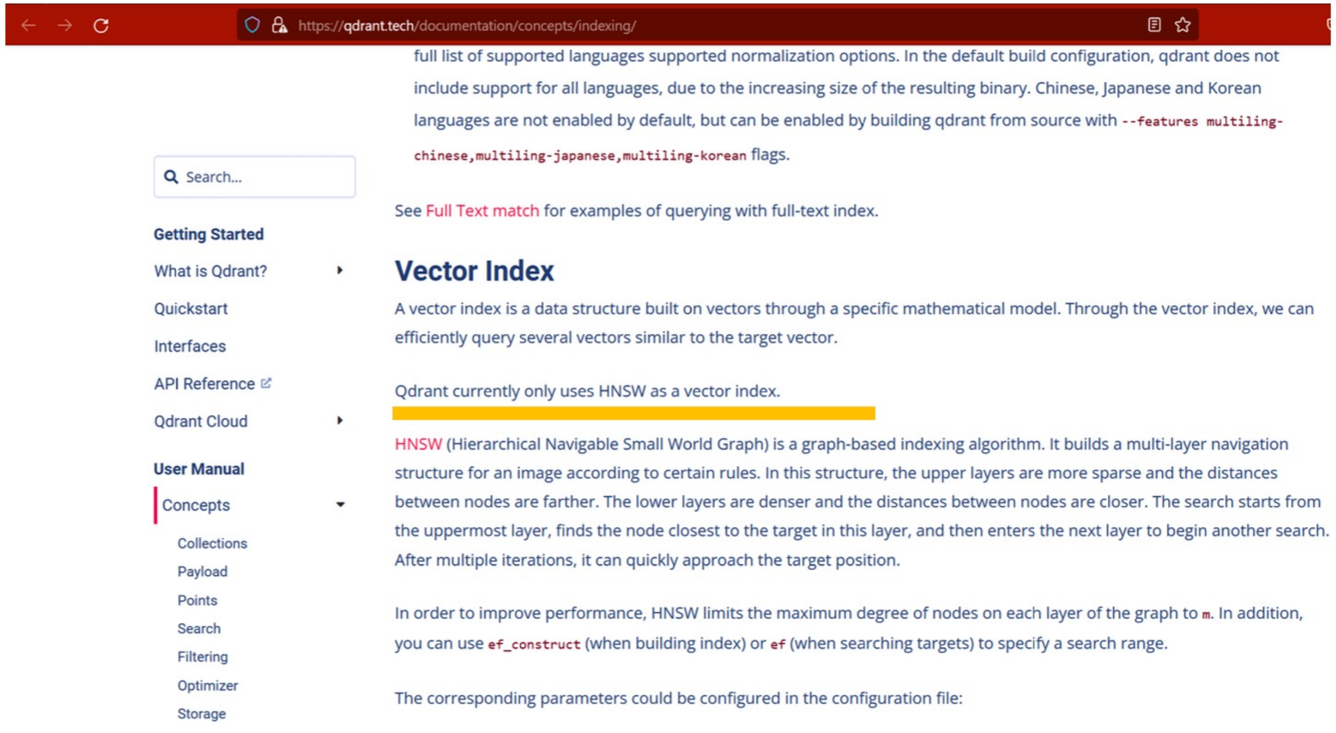
In this section, we will explore an algorithm for Approximate Nearest Neighbors, called **Hierarchical Navigable Small Worlds (HNSW)**.



HNSW in the real world

It is the same algorithm that powers **Qdrant**, the open-source Vector DB used by **Twitter's** (X) **Grok** LLM, which can access tweets in real time.





HNSW: Idea #1

HNSW is an evolution of the **Navigable Small Worlds** algorithm for Approximate Nearest Neighbors, which is based on the concept of **Six Degrees of Separation**.

Milgram's experiment aimed to test the social connections among people in the United States. The participants, who were initially located in Nebraska and Kansas, were given a letter to be delivered to a specific person in Boston. However, they were not allowed to send the letter directly to the recipient. Instead, they were instructed to send it to someone they knew on a first-name basis, who they believed might have a better chance of knowing the target person.

At the end of Milgram's small-world experiment, Milgram found that most of the letters reached the final recipient in five or six steps, creating the concept that people all over the world are connected by six degrees of separation.

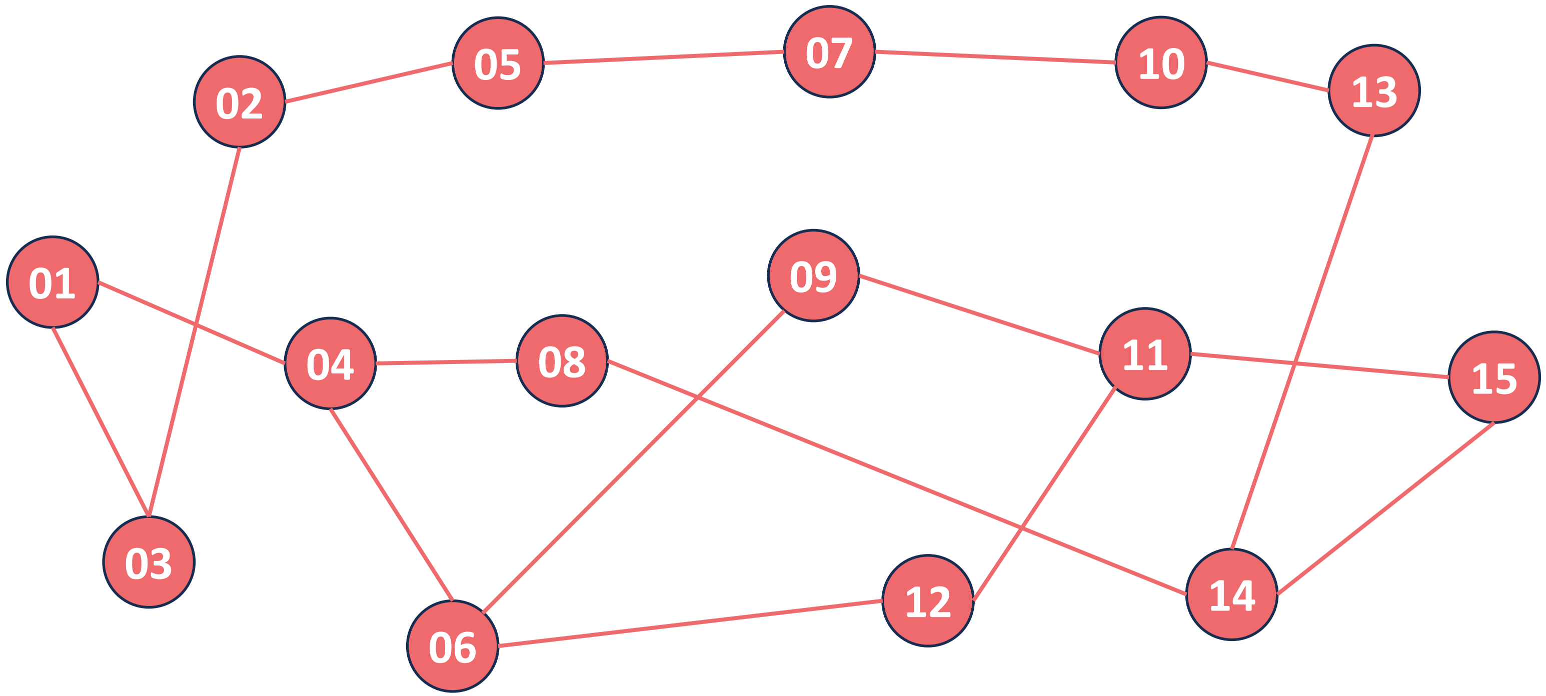
Facebook found in 2016 that its 1.59 billion active users were connected on average by 3.5 degrees of separation: Link

This means that you and Mark Zuckerberg are only 3.5 connections apart!

Navigable Small Worlds

The NSW algorithm builds a graph that — just like Facebook friends — connects close vectors with each other but keeping the total number of connections small. For example, every vector may be connected to up to 6 other vectors (to mimic Six Degrees of Separation).

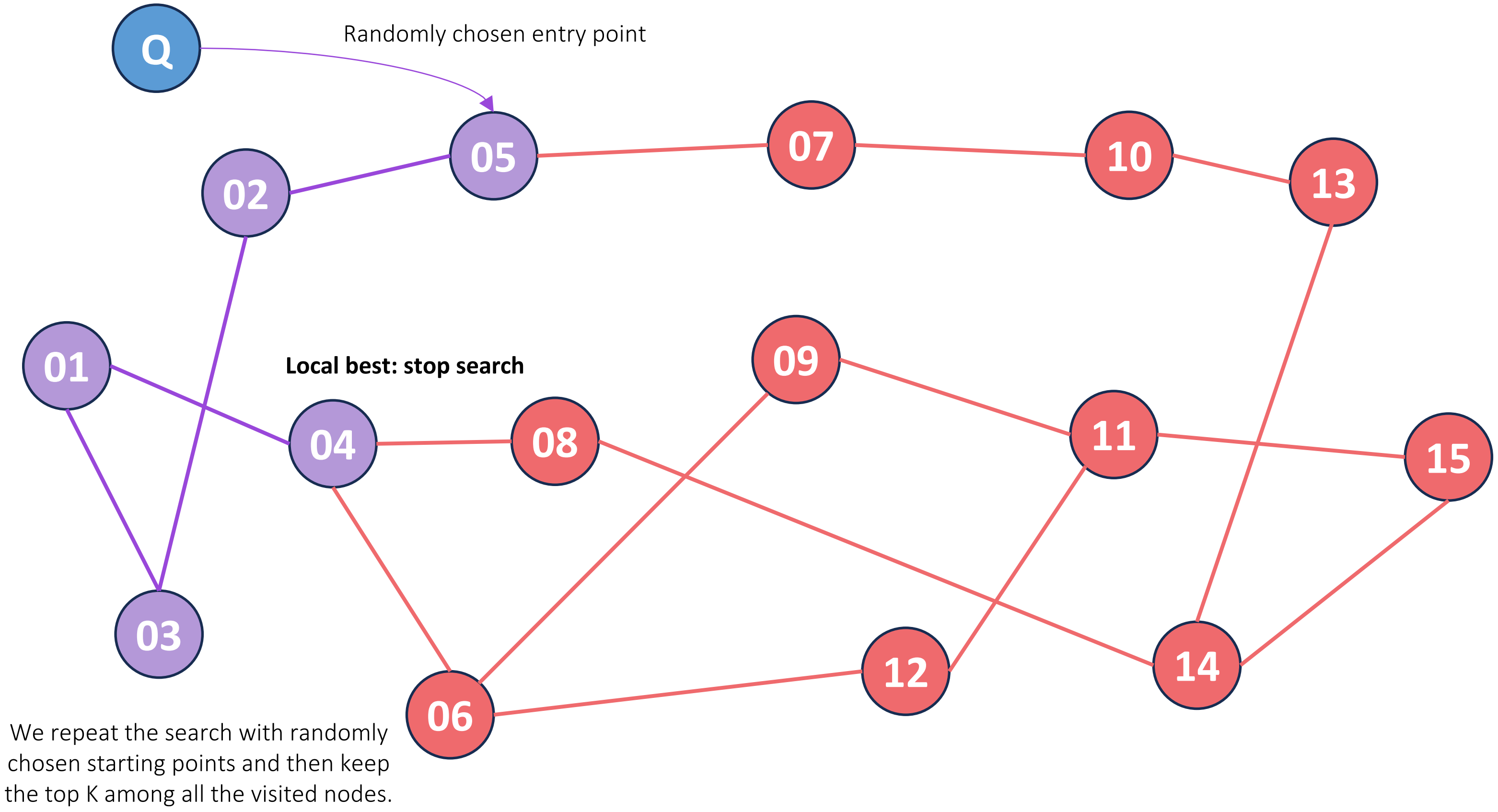
Node	Text
01	[...] The Transformer is a model [...]
02	[...] Diagnose cancer with AI [...]
03	[...] A transformer-based model [...]
04	[...] The Transformer has 6 layers [...]
05	[...] An MRI machine that costs $1 [...]
06	[...] The dot-product is a [...]
07	[...] Big-Pharma is not so big [...]
08	[...] Cross-Attention is a great [...]
09	[...] To solve an ODE [...]
10	[...] We are aging too fast [...]
11	[...] Open-source models like [...]
12	[...] MathBERT: A new model [...]
13	[...] AI to control aging [...]



Navigable Small Worlds: Searching for K-NN

Given the following query: "How many Encoder layers are there in the Transformer model?"
How does the algorithm find the K Nearest Neighbors?

Node	Text
01	[...] The Transformer is a model [...]
02	[...] Diagnose cancer with AI [...]
03	[...] A transformer-based model [...]
04	[...] The Transformer has 6 layers [...]
05	[...] An MRI machine that costs $1 [...]
06	[...] The dot-product is a [...]
07	[...] Big-Pharma is not so big [...]
08	[...] Cross-Attention is a great [...]
09	[...] To solve an ODE [...]
10	[...] We are aging too fast [...]
11	[...] Open-source models like [...]
12	[...] MathBERT: A new model [...]
13	[...] AI to control aging [...]

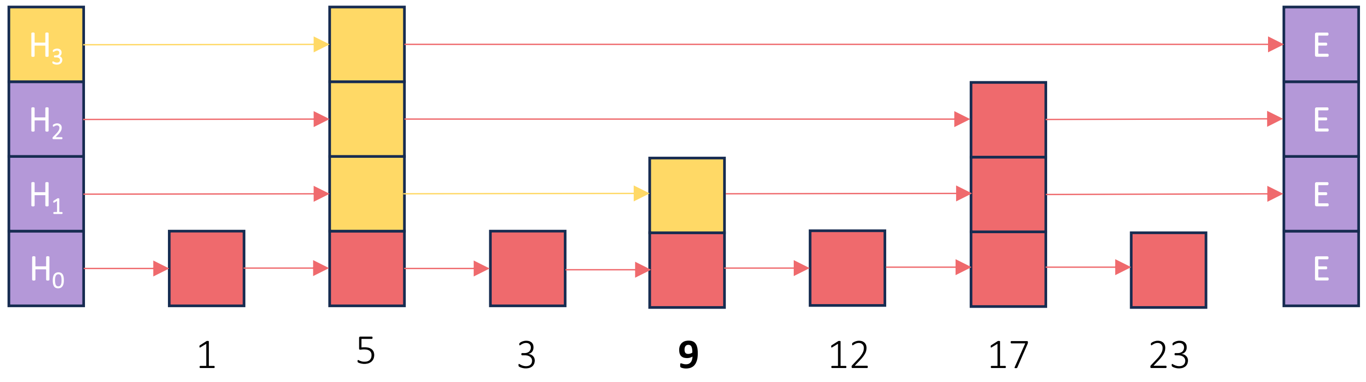


HNSW: Idea #2

To go from NSW (Navigable Small Worlds) to HSNW (Hierarchical SNW), we need to introduce the algorithm behind the data structure known as **Skip-List**.

The skip list is a data structure that maintains a sorted list and allows search and insertion with an average of $O(\log N)$ time complexity.

Let's search for the number **9**.

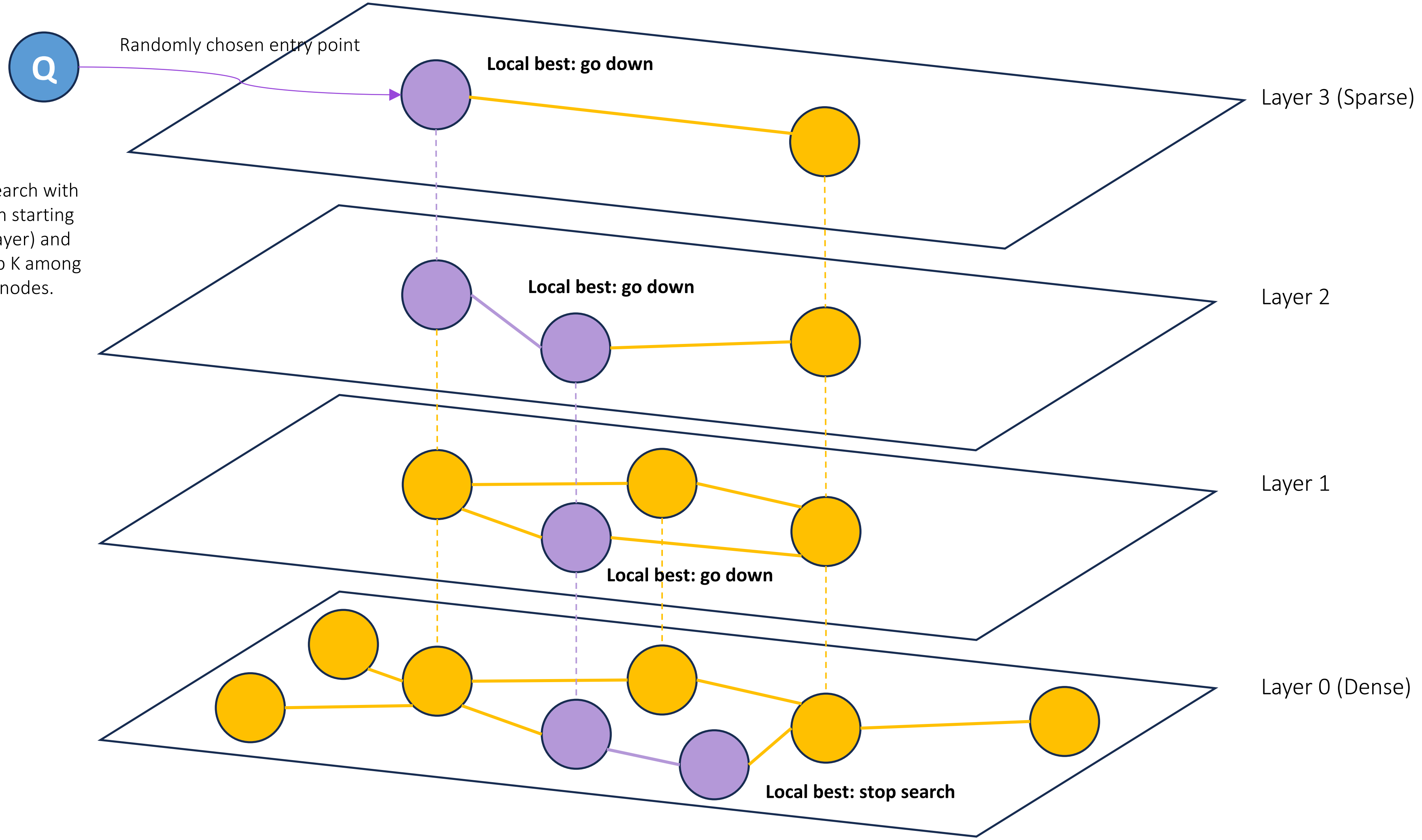


HNSW: Hierarchical Navigable Small Worlds

HNSW: Hierarchical Navigable Small Worlds

Let's search!

We repeat the search with randomly chosen starting points (on top layer) and then keep the top K among all the visited nodes.



QA with Retrieval Augmented Generation